PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applicant: Mark Robert Funk et al.

Application: **METHOD, APPARATUS AND COMPUTER PROGRAM PRODUCT FOR IMPLEMENTING BREAKPOINT BASED PERFORMANCE MEASUREMENT**

Serial No.: 10/616,525

Filing Date: July 10, 2003

Art Unit: 2192

Examiner: J. Derek Rutten

Case: ROC920020205US1

APPEAL BRIEF FOR APPLICANTS

JOAN PENNINGTON
535 North Michigan Avenue
Unit 1804
Chicago, Illinois 60611

One of the Attorneys for Applicants

TABLE OF CONTENTS

	Page
(1) REAL PARTY IN INTEREST.	5
(2) RELATED APPEALS AND INTERFERENCES.	5
(3) STATUS OF CLAIMS	5
(4) STATUS OF AMENDMENTS	5
(5) SUMMARY OF CLAIMED SUBJECT MATTER	5
(6) GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL	21
(7) ARGUMENT.	22
A. INTRODUCTION	22
B. THE SCOPE AND CONTENT OF THE PRIOR ART.	22
C. THE REJECTION OF CLAIMS 1-2, 4-6, 8-11, 13 and 14 UNDER 35 USC 103 SHOULD BE REVERSED	25
Claim 1 is patentable	26
Claim 4 is patentable	35
Claim 6 is patentable	37
Claim 11 is patentable	40
D. THE REJECTION OF CLAIM 7 UNDER 35 USC 103 SHOULD BE REVERSED	44
Claim 7 is patentable	44
E. CONCLUSION.	45
(8) CLAIMS APPENDIX	46
(9) EVIDENCE APPENDIX	52
(10) RELATED PROCEEDINGS APPENDIX.	53

TABLE OF CITATIONS

	Page
Arkie Lures, Inc. v. Gene Larew Tackle, Inc. 119 F.3d 953, 957, 43 USPQ2d 1294, 1297 (Fed. Cir. 1997)	31
Boehringer Ingelheim Vetmedica, Inc. v. Schering-Plough Corp., 320 F.3d 1339, 1347 (Fed. Cir. 2003)	42
Carl Schenck, A.G. v. Nortron Corp. 713 F.2d 782, 218 USPQ 698 (Fed. Cir. 1983)	31
CCS Fitness, Inc. v. Brunswick Corp., 288 F.3d 1359, 1366 (Fed. Cir. 2002)	42
In re Demiski 796 F.2d 236,443, 230 USPQ2d 313, 316 (Fed. Cir. 1986) ..	29
In re John R. Fritch 972 F.2d 1260, 23 USPQ2d 1780 (Fed. Cir. 1992)	34, 35
In re Gordon and Sutherland 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1983)	32, 34
Graham v. John Deere 383 U.S. 1, 148 USPQ 459, (1966)	26, 34
In re Gurley 27 F.3d 551, 553, 31 USPQ2D 1130, 1131 (Fed. Cir. 1994) . .	33
In re Kotzab 217 F.3d 1365, 1370, 55 USPQ2d 1313, 1317 (Fed. Cir. 2000)	31
In re Oetiker, 977 F.2d 1443, 24 USPQ2D 1443 (Fed. Cir. 1992)	31
ResQNet.com, Inc. v. Lansa, Inc., 346 F.3d 1374, 1378 (Fed. Cir. 2003).	42
In re Sernaker 702 F.2d 989, 217 USPQ 1 (Fed. Cir. 1983)	34
Vitronics Corp. v. Conceptronic, Inc., 90 F.3d 1576, 1582 (Fed. Cir. 1996)	43
W.L. Gore & Assocs. Inc. V. Garlock, Inc. 721 F.2d 1540, 1553, 220 USP1 303, 313 (Fed. Cir. 1984), cert. denied, 469 U.S. 851 (1984)	29
In re Young 927 F.2d 588, 18 USPQ2d 1089 (Fed. Cir. 1991)	31

TABLE OF OTHER AUTHORITIES

35 U.S.C. §103.	26

PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applicant: Mark Robert Funk et al.

Application: **METHOD, APPARATUS AND COMPUTER PROGRAM PRODUCT FOR IMPLEMENTING BREAKPOINT BASED PERFORMANCE MEASUREMENT**

Serial No.: 10/616,525

Filing Date: July 10, 2003

Art Unit: 2192

Examiner: J. Derek Rutten

Case: ROC920020205US1

---

535 North Michigan Avenue
Unit 1804
Chicago, Illinois 60611

Mail Stop **Appeal Brief Patents**
Honorable Commissioner Of Patents
P.O Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF FOR APPLICANTS

Sir:

This is an appeal of the final rejections of all the claims 1-2, 4-11, 13 and 14 in the office action made final and mailed December 28, 2007. For the reasons set forth below, it is submitted that the Board should reverse the final rejections of claims 1-2, 4-11, 13 and 14.

(1) REAL PARTY IN INTEREST

The real party of interest is International Business Machines Corporation.

(2) RELATED APPEALS AND INTERFERENCES

Applicants' attorney knows of no other appeals or interferences that would have a bearing on the Board's decision in the present appeal.

(3) STATUS OF CLAIMS

Claims 1-2, 4-11, 13 and 14 have been finally rejected under 35 U.S.C. § 103(a) in an office action mailed December 28, 2007. The rejection of each of the claims 1-2, 4-11, 13 and 14 has been appealed. Claims 3, and 12 have been canceled.

(4) STATUS OF AMENDMENTS

A request for reconsideration without amendment of the claims was filed after the final rejection.

(5) SUMMARY OF CLAIMED SUBJECT MATTER

The claimed invention as recited by independent claims 1, 6, and 11, and representative separately patentable dependant claims 4, and 7, can best be appreciated and understood with reference to the patent specification (hereinafter page p., line l.).

The present invention overcomes problems of the prior art for implementing performance measurement. In software and compiler development, it is often needed to determine the amount of time needed to execute some function. It is also of interest to determine how often various hardware functions are executed within this function. A common technique currently used to measure the time during a function

is to request a time stamp both before and after the function and subtract the two in order to determine the elapsed time. With the use of time stamps merely asking for the start and end times takes time, which can affect the results. Similarly, during the processing of the request for counts of other hardware functions results in those same hardware functions continuing to occur during the requests, which also can effect the results. Ideally, the timing and counting of hardware functions would be limited to only the function being tested. One possible solution is to repetitively execute the function in an attempt to amortize the measurement cost over the entire period of the test. A problem with this measurement technique, however, is the time required for repetitively executing the function. (p. 1, l. 9 - p. 2, l. 1)

The present invention enables improved timing of a particular function, and to count selected programmable events during the execution of a function with a breakpoint manager and performance measurement program in accordance with the preferred embodiment. The present invention enables programmable processor event counters to start counting immediately at the beginning of this bounded call flow and stop counting immediately at the end of the bounded call flow. The bounds of the call flow are provided at arbitrary locations within the instruction stream. (p. 6, l. 1 - 6, 13 - 34; p. 7, l. 14 - 20)

The present invention provides an admittedly novel method, apparatus and computer program product for implementing breakpoint based performance measurement. (p. 2, l. 15 - 27)

Independent claim 1 recites a method for implementing breakpoint based performance measurement using a set of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and cache misses; said method comprising:

providing compiler-generated hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction;

inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code;

executing said compiler-generated hardware instructions and suspending processing of said hardware instructions responsive to executing said start breakpoint instruction;

responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions;

said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction;

executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction; and

providing a debugger breakpoint manager including a performance measurement program and a user interface, and enabling a user to specify a start bound and an end bound of a performance collection region of said user source code and said set of hardware counters. (p. 2, l. 12 - 27, p. 4, l. 1 -2 1; p. 5, l. 3- 22; p. 6, l. 13 - 34; p. 7, l. 14 - p. 9, l. 7)

In accordance with features of the invention, a method is provided to determine the performance characteristics of a single call flow executed by a single task, rather than counting events within the overall operating system. The instructions within the single call flow are bounded to describe the period over which the programmable processor event counters 140 are incremented. The programmable processor event counters 140 start counting immediately at the beginning of this bounded call flow and stop counting immediately at the end of the bounded call flow. The bounds of the call flow are provided at arbitrary locations within the instruction stream. (p. 5, l. 3 - 12)

Dependent claim 4 recites a method for implementing breakpoint based performance measurement as recited in claim 1 further defines the inserting step includes inserting said start breakpoint instruction and said stop breakpoint instruction at arbitrary user defined locations in said hardware instructions. (p. 5, l. 3 - 12; p. 7, l. p. 7, l. 14 - 31)

Independent claim 6 recites apparatus for implementing breakpoint based performance measurement comprising:

a plurality of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events;

said predefined programmable processor events including processor cycles and cache misses;

a source level debugger including a breakpoint manager;

said breakpoint manager including a performance measurement program and a user interface;

said breakpoint manager utilizing said performance measurement program and said user interface for defining a set of said hardware counters for counting user specified programmable processor events and for inserting a start breakpoint instruction and a stop breakpoint instruction in hardware instructions;

said breakpoint manager enabling a user to specify a start bound and an end bound of a performance collection region of a user source code and said set of hardware counters;

a compiler generating hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction and inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code;

user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions responsive to executing said start breakpoint instruction and generating a processor interrupt for entering interrupt handler instructions and for calling said breakpoint manager;

said breakpoint manager for generating a start processing instruction to return

processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction; and

said user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction. (p. 2, l. 12 - 27, p. 4, l. 1 -2 1; p. 5, l. 3- 22; p. 6, l. 13 - 34; p. 7, l. 14 - p. 9, l. 7)

The present invention provides code or breakpoint handler instructions that allows the user of the debugger 134 to interrogate the program state and most importantly allows the user to request a return to program processing. The breakpoint handler does this by ultimately executing a single instruction indicated at a line labeled START PROCESSING that returns processing out of the interrupt handler and into the program again, immediately after the point of the first breakpoint interrupt, as indicated at a line labeled START RETURN-FROM-INTERRUPT INSTRUCTION from interrupt handler instructions 206 to the compiler-generated instructions 204. The reason this is important to the breakpoint performance measurement method of the preferred embodiment is that at this point in time, the initialized programmable hardware counters 140 are enabled to begin counting. It is this point in time that defines the beginning of the bounded instruction stream. Now the user's program is executing, calling arbitrary routines, using arbitrary processor facilities, and counting arbitrary processor events. At some point in time execution reaches the ending bound of the measurement period.

The value in a breakpoint instruction in accordance with the preferred embodiment essentially selects which of the possible types of events are to be counted with the programmable processor event counters. (p. 7, l. 14 - p. 8, l. 14)

Dependent claim 7 recites apparatus for implementing breakpoint based performance measurement as recited in claim 6 wherein start breakpoint instruction includes encoded information specifying said defined set of hardware counters. (p. 5, l. 14 - 33; p. 6, l. 13 - 34)

Independent claim 11 recites a computer program product for implementing breakpoint based performance measurement in a computer system, said computer program product including instructions executed by the computer system to cause the computer system to perform the steps of:

defining a set of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and translation lookaside buffer misses;

providing compiler-generated hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction;

inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code;

executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions responsive to executing

said start breakpoint instruction;

responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions;

said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction;

executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction; and

receiving user selections for a start bound and an end bound of a performance collection region of said user source code and said set of hardware counters. (p. 2, l. 12 - 27, p. 4, l. 1 -2 1; p. 5, l. 3- 22; p. 6, l. 13 - 34; p. 7, l. 14 - p. 9, l. 7)

In FIGS. 1A and 1B there is shown a computer system generally designated by the reference character 100 for implementing breakpoint based performance measurement in accordance with the preferred embodiment. Computer system 100 includes a main processor 102 or central processor unit (CPU) 102 coupled by a system bus 106 to a memory management unit (MMU) 108 and system memory including a dynamic random access memory (DRAM) 110, a nonvolatile random access memory (NVRAM) 112, and a flash memory 114. The system bus 106 may be private or public, and it should be understood that the present invention is not limited to a particular bus topology used. A mass storage interface 116 coupled to the system bus 106 and MMU

108 connects a direct access storage device (DASD) 118 and a CD-ROM drive 120 to the main processor 102. Computer system 100 includes a display interface 122 connected to a display 124, and a network interface 126 coupled to the system bus 106. (p. 3, l. 11 - 25)

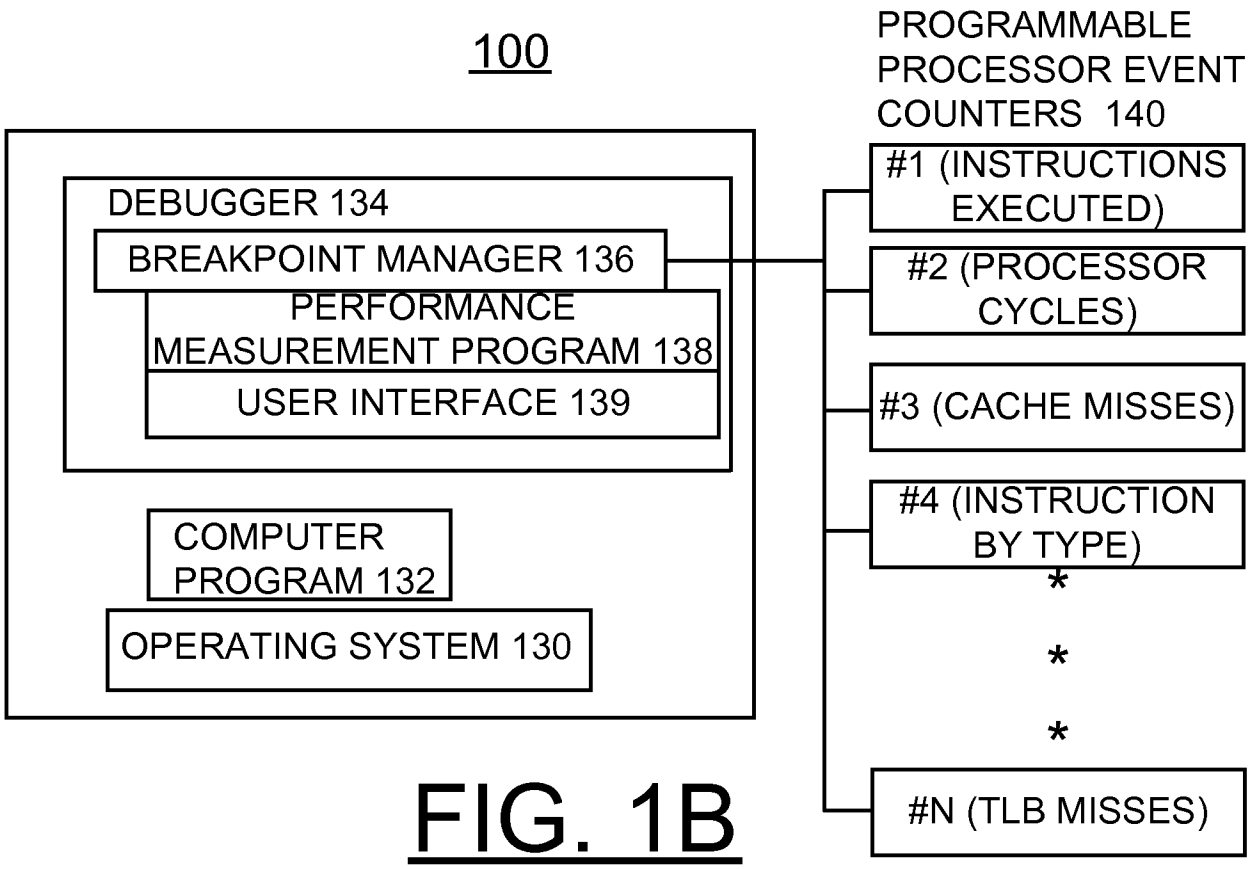

FIG. 1B

As shown in FIG. 1B, computer system 100 includes an operating system 130, a computer program 132, and a source level debugger 134 including a breakpoint manager 136, a performance measurement program 138 of the preferred embodiment, and a user interface 139. A plurality of programmable processor event counters 140,

#1-#N are coupled to and operatively controlled by the breakpoint manager 136 with the performance measurement program 138 in accordance with the preferred embodiment. The programmable processor event counters 140, #1-#N in some embodiments are included in processor 102 and are programmable to count various types of events recognized by the processor 102. (p. 4, l. 1 - 10)

A common debugging technique for the debugger 134 is used to install breakpoints into code in order to allow for a function to cease processing at that breakpoint and present the state of the program at that point. The low level support for this breakpoint is essentially a hardware instruction resulting in an interrupt out of executing code and processing being routed to the debugger. When the tool user decides to continue processing, the debugger arranges for the instruction overlaid by the breakpoint instruction to be executed and then executes a return from interrupt instruction in order to return to the processing of the interrupted function. Processing continues until the next breakpoint instruction is executed. (p. 4, l. 11 - 21)

Events that can be counted by the programmable processor event counters 140, #1-#N include, for example, a number of instructions executed, a number of processor cycles spent executing a function (i.e., amount of time), a number of cache misses, a number of particular type of instructions (e.g. conditional branch instructions), a number of translation lookaside buffer (TLB) misses, and the like. Coupled with their capability of producing a processor interrupt when the programmable processor event counters 140 overflow, the programmable processor event counters 140 are typically used to provide a profile of how various software components of the operating system

use the processor 102. In this mode the programmable processor event counters 140 tend to be free running and independent of a task, also known as a thread, or program that is executing. Typically, such as in the Power PC AS processor, the number of counters 140 is far less than the number of types of events that can be counted. For this reason, counters 140 are programmable for the type of performance analysis required. (p. 4, l. 22 - p. 5, l. 2)

In accordance with features of the preferred embodiment, the source level debugger 134 is used to define bounds within an instruction stream. The debugging tool's breakpoint support is used in presenting an elapsed time over an operation or function executed between two breakpoints. Along with timing of a particular function, it is also useful to be able to count selected programmable events during the execution of a function with the breakpoint manager 136 and performance measurement program 138 in accordance with the preferred embodiment. The value in a breakpoint instruction in accordance with the preferred embodiment essentially selects which of the possible types of events are to be counted with the programmable processor event counters 140. (p. 5., l. 13 - 23)

Processor 102, such as a PowerPC processor and various other processors, support multiple programmable counters 140, where sets of these counters can count various types of events in the hardware ranging from cycles, to instructions, to cache misses, to execution of various classes of instructions. (p. 5., l. 29 - 33)

In accordance with features of the preferred embodiment, with hardware counter support, this debugger breakpoint tool becomes even more useful by limiting the

measurement to strictly the function between the breakpoints. The measurement is limited only the code that executes between the breakpoints. This is from the moment that the code begins execution to the moment that the next breakpoint interrupt occurs. Without it there is a fair amount of overhead outside of this measured code. This overhead further implies that the length of the performance run needs to be large enough so that the overhead becomes inconsequential. For instance, the best that we can do to measure a snippet of code is to materialize the time-of-day in the code both before and after the snippet. But even this takes some considerable time. (p. 6, l. 1 - 12)

In accordance with features of the preferred embodiment, two instructions, an end breakpoint instruction and a start processing instruction, are used. The end breakpoint instruction is used that has the effect of interrupting and shutting down or freezing a set of selected counters 140. This function of this instruction is used at the end of a measured period. Prior to the beginning of the measured period a similar start breakpoint instruction informs the breakpoint manager 136 of the types of counters 140 to be included within the measured period via encode bits of the instructions. (p. 6, l. 13 - 20)

In accordance with features of the preferred embodiment, a set of bits in the first breakpoint instruction can be used to define for the breakpoint manager 136 not for the hardware, a set of hardware counters 140 which are to be used to count hardware events, such as execution cycles (i.e. time). The breakpoint manager 136 identifies for the hardware those counters 140 that are to be used, initializes those counters 140, and

indicates to the hardware that counting will begin with a trigger event. The trigger event here is really the execution of the start processing instruction. The counters would continue until processing reaches the next breakpoint instruction. The breakpoint manager 136 interrogates the previously programmed counters 140, save them for subsequent performance analysis. If the breakpoint instruction indicates that the counters should again be used, it starts the process over again. If the breakpoint instruction indicates that no counters are to be used, this process has effectively ended. (p. 6, l. 21 - 34)

The start processing instruction is a return-from-interrupt instruction that starts the selected counters 140 enabled by the breakpoint manager 136 as specified in the preceding breakpoint instruction. The start processing or return-from-interrupt instruction is the last instruction of the breakpoint manager 136 and the first instruction of the code to be measured. For example, in the PowerPC AS architecture, these instructions can be provided by using some of the currently unused encodes of the existing system call/system call vectored (SC/SCV) and return from interrupt and return from system call vectored (RFID/RFSCV) instructions. With this enhanced support, the user is aware of other performance perturbing effects, such as, page faults, TLB misses, and cache misses, and that a task switch had occurred during the measurement period from the selected counters 140 enabled by the breakpoint manager 136. (p. 7, l. 1 - 13)

FIG. 2 there is shown an exemplary logical implementation generally designated by the reference character 200 for breakpoint based performance measurement in

accordance with the preferred embodiment. A user of the source level debugger 134 defines locations in the instruction stream as shown by start and end of a performance collection region in a source code 202 where the source level debugger 134 temporarily installs. (p. 7, l. 14 - 20)

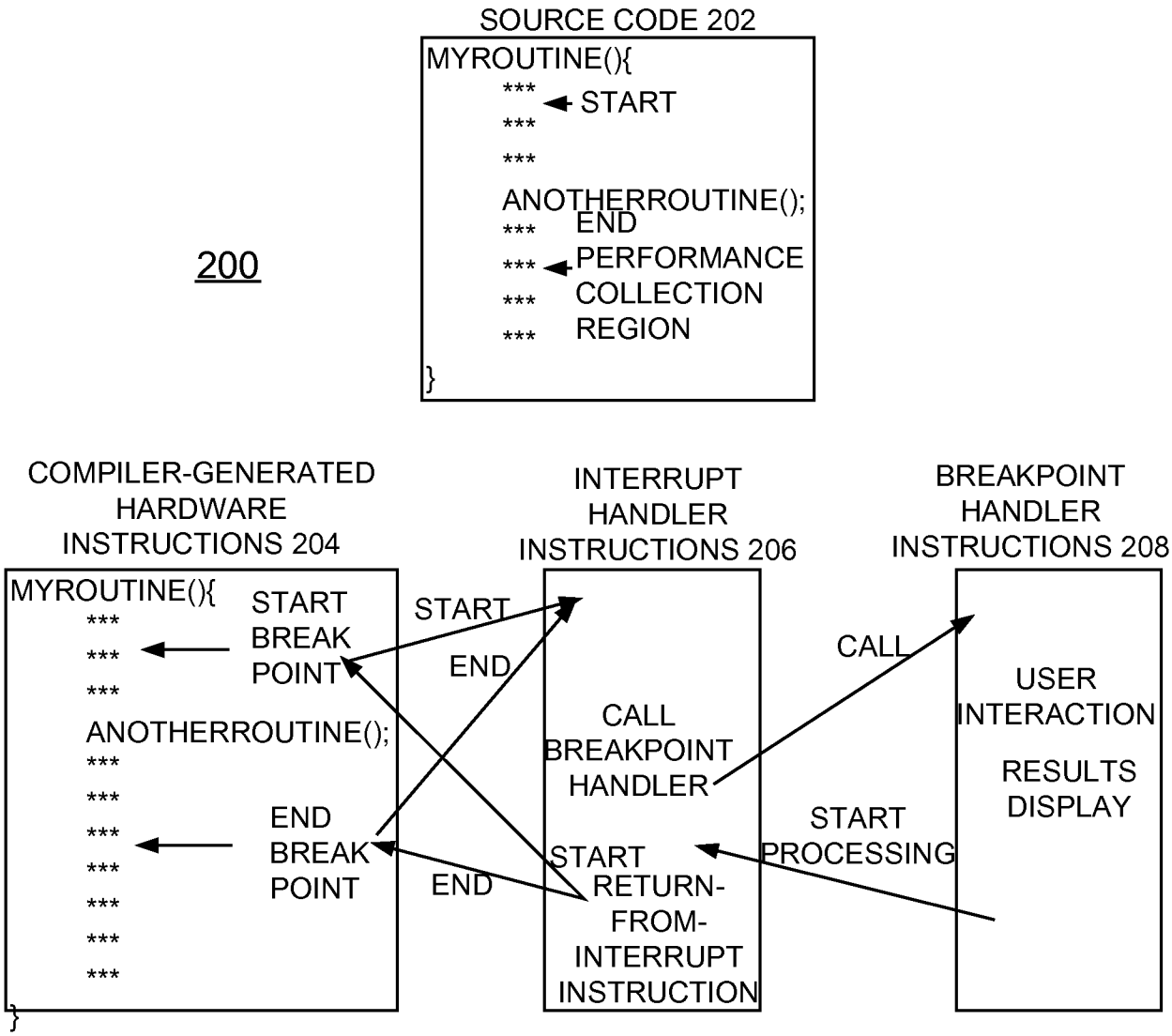

FIG. 2

Compiler-generated hardware instructions 204 define breakpoint instructions within the instruction stream. Upon execution of a program, processing of that program is temporarily suspended when the breakpoint instruction is executed. The reason that program is temporarily suspended is that the breakpoint instruction produces a processor interrupt. An interrupt saves the processor state at the point of the breakpoint, enters a completely separate instruction stream of interrupt handler instructions 206 at the location of an interrupt handler, as indicated at a line labeled START from the compiler-generated instructions 204 to the interrupt handler instructions 206, and ultimately enters the code of breakpoint handler instructions 208 associated with a breakpoint handler, as indicated at a line labeled CALL. (p. 7, l. 21 - 31)

In accordance with features of the preferred embodiment, this code or breakpoint handler instructions 208 allows the user of the debugger 134 to interrogate the program state and most importantly allows the user to request a return to program processing. The breakpoint handler does this by ultimately executing a single instruction indicated at a line labeled START PROCESSING that returns processing out of the interrupt handler and into the program again, immediately after the point of the first breakpoint interrupt, as indicated at a line labeled START RETURN-FROM-INTERRUPT INSTRUCTION from interrupt handler instructions 206 to the compiler-generated instructions 204. The reason this is important to the breakpoint performance measurement method of the preferred embodiment is that at this point in time, the initialized programmable hardware counters 140 are enabled to begin counting. It is this point in time that defines the

beginning of the bounded instruction stream. Now the user's program is executing, calling arbitrary routines, using arbitrary processor facilities, and counting arbitrary processor events. At some point in time execution reaches the ending bound of the measurement period. (p. 7, l. 32 - p. 8, l. 14)

In accordance with features of the preferred embodiment, this ending bound is also defined by an interrupt due to the execution of a breakpoint instruction, again optionally installed by the source level debugger, labeled BREAKPOINT in the compiler-generated hardware instructions 204. It is at this point in time that the programmable hardware counters 140 stop counting. It is at exactly this point in time, not at some point later within the execution of the breakpoint handler. Doing otherwise would result in the counting of some unknown number of events outside of the function being measured. Although possible, creating a tool that counts events, and this includes time, outside of the measurement period would decrease the usefulness of the tool. It is the coupling of these performance counters 140 with the precision of starting and stopping of the counters 140 via interrupt-state transitioning instructions that is the core of the breakpoint performance measurement method of the preferred embodiment. (p. 8, l. 15 - 28)

In addition to simply starting and stopping the counters, the user is able to specify, via the debugger breakpoint manager 136 including the performance measurement program 138 and user interface 139, that the arbitrary instruction stream is bounded in this way and to specify the types of events that the counters should count. Along with specifying via the source level debugger 134 the bounds of the

measurement period, the user also specifies the type of events to be counted. An encoded version of this information can be carried in the breakpoint instruction 204 representing the beginning of the measurement period. An alternative would be for the debugger to record this information and the task ID on the user's thread prior to executing the breakpoint instruction 204 starting the measurement period. One straightforward way of doing this would be to first define the bounds, start the program to execute the time of the first of the breakpoints, and then have the debugger request the needed information. (p. 8, l. 29 - p. 9, l. 7).

FIG. 3, an article of manufacture or a computer program product 300 of the invention is illustrated. Recording medium 302 stores program means 304, 306, 308, 310 on the medium 302 for carrying out the methods for implementing breakpoint based performance measurement of the preferred embodiment in the system 100 of FIGS. 1A and 1B. (p. 9, l. 8 - 17)

(6) GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Two grounds of rejection presented for review are:

Whether claims 1-2, 4-6, 8-11, 13 and 14 are unpatentable under 35 USC §103(a) over Bickle et al. publication entitled "Differential Effective Lapse Time Accumulator (Delta) in view of "How Debuggers Work" by Rosenberg in view of Dreyer et al., U.S. Patent 5,657,253, in view of Carter et al., U.S. Patent 6,249,907.

Whether claim 7 is unpatentable under 35 USC §103(a) over Bickle and Rosenberg, Dreyer and Carter further in view of Hawley et al, U.S. patent 5,533,192.

(7) ARGUMENT

A. INTRODUCTION

Applicants respectfully submit that the Examiner's rejections of all pending claims 1-2, 4-11, 13 and 14 under 35 U.S.C. § 103(a) should be reversed because the subject matter of each of the independent claims 1, 6, and 11 and the representative dependant claims 4, and 7 is patentable over all the references of record. There is no teaching or suggestion in any of the cited references, individually or taken as a whole, to make the claimed invention obvious. The rejections of all of the pending claims 1-2, 4-11, 13 and 14 under 35 U.S.C. §103(a) are improper and should be reversed.

B. THE SCOPE AND CONTENT OF THE PRIOR ART

The Bickle et al. publication entitled Differential Effective Lapse Time Accumulator (DELTA) discloses a test tool, DELTA, applied to processors whose address bus and control signals are defined as conditions and timing measurements are taken between address (breakpoint) A and address B or to count the number of times the processor executes a particular bus state. The tool allows the measurement of system performance in two areas: (1) instruction cycle time measurement and (2) program execution time, instruction count and busy state time measurement, under the control of a microprocessor 12. The Bickle et al. publication discloses a display terminal 25 providing an operator interface that is used to enter breakpoint and oscillator parameters and to display timing/counting results under the control of the microprocessor 12.

"How Debuggers Work" by Rosenberg discloses hardware debugger facilities, at

pages 39-40 states the minimum basic requirements a debugger place on underlying hardware are 1. A way to specify a breakpoint such that when the processor reaches this location, execution will stop. 2. A notification system, also called an interrupt or a trap, that will notify the operating system (and thereby the debugger) that an important event has occurred with respect to the running process. 3. The ability to read and write directly out of and into the hardware registers when the interrupt occurs; this includes the program counter register. At page 40, Rosenberg describes Breakpoint Support, where breakpoints are implemented as a special instruction that causes a trap to the operating system, which then can notify a special program that has registered itself as a debugger. At page 41, Rosenberg describes the debugger writing a special breakpoint instruction.

Dreyer et al., U.S. Patent 5,657,253 discloses an apparatus for measuring and monitoring various parameters that contribute to the performance of a processor that includes a pair of programmable event counters for counting any two independent events selected from a predetermined list of processor events. A specialized register controls the operation of the event counters and also selects the events to be counted. The contents of the event counters can be accessed either by a supervisor mode program which reads an instruction or through a special access port.

Carter et al., U.S. Patent 6,249,907 discloses a system for debugging a computer program. A user indicates a specified breakpoint type, such as a program statement, variable reference, command, and the like. The program, including program statements, is then compiled. During compilation, the compiler locates statements in

the program corresponding to the breakpoint types and generates a function call into the program at instances in the program of statements corresponding to the user specified breakpoint types. During a debugging phase, a debugger may execute an executable version of the program, including the function calls. Upon processing the function calls, the debugger may stop execution of the program and pass control to the user to perform debugging operations. As stated at column 4, lines 35-52 and column 5, lines 4-15:

> The debugger 16 and program when executing may communicate via the operating system 12. In the embodiments where the debugger 16 and the program execute in separate processes, the debugger 16 can control the execution of the object code 20 or obtain information on the execution of the object code 20 via system calls, i.e., APIs, to the operating system 12, which in turn directly controls the execution of the object code 20. In the embodiments where the debugger 16 and object code 20 execute within the same process, calls to the debugger 16 can be placed in the object code 20 so that during the execution of the object code 20, the object code 20 can call, via the operating system 12, subroutines of the debugger 16 to pass control to the debugger 16. Preferred embodiments of the present invention may be implemented in either environment, i.e., the execution of the debugger 16 and object code 20 may be in the same or different processes.
> * * *
> During compilation, the compiler 14 will generate breakpoint indicators or "hooks" into the object code 18 at each instance in the program of the statement corresponding to the user specified breakpoint type. Hooks are instructions the compiler 14 inserts into the program, usually the object code 20, during compilation. The hooks can be used to set breakpoints that instruct the debugger 16 to gain control of the program at specific points in the program. The hooks may be inserted at the entrances and exits of blocks, at statement boundaries, and at points in the program where program flow might change, such as before and after a procedure call.

Hawley et al, U.S. patent 5,533,192 discloses a program debugging system having a core unit that includes a plurality of debugger memory areas, each uniquely associated with a corresponding one of a plurality of debuggers. The core unit responds to an exception condition by selecting one debugger from the plurality of

debuggers, selection being made by determining which one of the debuggers is associated with the program exception. Then, computer state information and debugger state information are stored into a selected one of the debugger memory areas that is exclusively associated with the selected debugger, and the selected debugger is activated. A new debugger may register with the core unit, so that the new debugger is added to the plurality of debuggers. The activated debugger may send a debugging command to the core unit, which responds by updating debugger state information based on the received debugging command, and storing the updated debugger state information into the selected debugger memory area. When a debugger relinquishes control of the computer, the core unit retrieves the updated debugger state information from the selected debugger memory area, and controls the hardware resources in accordance therewith. If the updated debugger state information includes an indication that a breakpoint is set, the core unit sets a breakpoint that includes information associating the set breakpoint with the selected debugger. When the breakpoint is triggered, the core unit identifies from the breakpoint information which of the debuggers the breakpoint is associated with, and activates the identified debugger.

C. THE REJECTION OF CLAIMS 1-2, 4-6, 8-11, 13 and 14 UNDER 35 USC 103 SHOULD BE REVERSED

The Board should reverse the rejection of the claims 1-2, 4-6, 8-11, 13 and 14 under 35 USC §103 over Bickle and Rosenberg, Dreyer and Carter.

Claim 1 is patentable

35 U.S.C. §103 requires that the invention as claimed be considered "as a whole" when considering whether the invention would have been obvious when it was made. Graham v. John Deere, 383 U.S. 1, 148 USPQ 459, 472 (1966). It is applicants' claimed invention which must be considered as a whole pursuant to 35 U.S.C. §103, and failure to consider the claimed invention as a whole is an error of law. The legal determination under section 103 is whether the claimed invention as a whole would have been obvious to a person of ordinary skill in the art at the time the invention was made.

Independent claim 1 recites a method for implementing breakpoint based performance measurement using a set of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and cache misses; said method comprising:

providing compiler-generated hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction;

inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code;

executing said compiler-generated hardware instructions and suspending processing of said hardware instructions responsive to executing said start breakpoint instruction;

responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions;

said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction;

executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction; and

providing a debugger breakpoint manager including a performance measurement program and a user interface, and enabling a user to specify a start bound and an end bound of a performance collection region of said user source code and said set of hardware counters.

Applicants respectfully that independent claim 1 is patentable over all the references of record. The total teachings of the cited references fail to render obvious the subject matter of the present invention as defined in independent claim 1.

The present invention provides enhanced breakpoint based performance measurement that is different from the prior art including the Bickle, Rosenberg, Dreyer et al., Carter et al., and Hawley.

Bickle, Rosenberg, Dreyer et al., Carter et al., and Hawley references fail to suggest inserting a start breakpoint instruction and a stop breakpoint instruction in hardware instructions; executing said hardware instructions and suspending processing

of said hardware instructions responsive to executing said start breakpoint instruction; and responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions, and that said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to the hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction, as taught and claimed by applicants.

Only applicants teach the subject matter of the invention, as recited by pending independent claim 1. The invention as claimed must be considered "as a whole" when considering whether the invention would have been obvious when it was made. The prior art references of record provide no teaching, suggestion or inference in the prior art as a whole or knowledge generally available to one having ordinary skill in the art to achieve the claimed invention.

Applicants only teach these steps of inserting a start breakpoint instruction and a stop breakpoint instruction in hardware instructions; executing said hardware instructions and suspending processing of said hardware instructions responsive to executing said start breakpoint instruction; and responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions, and that said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to the hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction.

The above recited steps define subject matter of the invention being claimed and must be considered by the Examiner in determining whether the subject matter would have been obvious.

Applicants respectfully submit that the combined teachings of the cited Bickle, Rosenberg, Dreyer, Carter and Hawley references disclose conventional testing and debugging mechanisms. However, Applicants respectfully submit that the cited Bickle, Rosenberg, Dreyer, Carter and Hawley references do not enable, nor suggest the subject matter of the invention as recited in the independent claim 1.

Contrary to the Examiner's assertion, Applicants respectfully submit that Rosenberg does not teach any "special hardware instruction", nor any equivalent hardware instruction. Rosenberg teaches conventional breakpoints that are implemented as a special instruction that causes a trap to the operating system, which then can notify a special program that has registered itself as a debugger.

Contrary to the Examiner's assertion, Carter does not disclose the claimed steps of the invention, nor any equivalent steps. To arrive at such a conclusion appears to require that the above expressly recited limitations set forth in claim 1 of the method of the invention are ignored or improperly incorporate applicants' teachings into Carter. Hindsight is impermissible when an examiner rejects an application in reliance upon teachings not drawn from any prior art disclosure, but from the applicant's own disclosure. In re Demiski, 796 F.2d 236,443, 230 USPQ2d 313, 316 (Fed. Cir. 1986); W.L. Gore & Assocs. Inc. V. Garlock, Inc., 721 F.2d 1540, 1553, 220 USP1 303, 313 (Fed. Cir. 1984), cert. denied, 469 U.S. 851 (1984).

The references of record do not suggest the claimed subject matter of claim 1, as is taught and claimed by Applicants. Only Applicants teach these features of such method of the invention.

The present invention provides a novel method which implements breakpoint based performance measurement. The claimed method is provided to determine the performance characteristics of a single call flow executed by a single task, rather than counting events within the overall operating system as in conventional arrangements.

Only Applicants teach inserting a start breakpoint instruction and a stop breakpoint instruction in hardware instructions. Only Applicants teach that said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to the hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction as recited in the independent method claim 1,

These limitations as recited in independent claim 1 are not disclosed nor inherent in the combined teachings of Bickle, Rosenberg, Dreyer, Carter and Hawley. Bickle, Rosenberg, Dreyer, Carter and Hawley fail to provide any equivalent steps. None of the references including Bickle, Rosenberg, Dreyer, Carter and Hawley disclose or achieve a method for implementing breakpoint based performance measurement as claimed.

A prima facie showing of obviousness under 35 U.S.C. §103 is established when the teachings from the prior art itself would appear to have suggested the claimed subject matter to a person of ordinary skill in the art. For a combination of prior art references to render an invention obvious, "[t]here must be some reason, suggestion, or

motivation found in the prior art whereby a person of ordinary skill in the field of the invention would make the combination." In re Oetiker, 977 F.2d 1443, 1447, 24 USPQ2D 1443, 1446 (Fed. Cir. 1992). It is insufficient to establish obviousness that the separate elements of the invention existed in the prior art, absent some teaching or suggestion, in the prior art, to combine the elements. Arkie Lures, Inc. v. Gene Larew Tackle, Inc., 119 F.3d 953, 957, 43 USPQ2d 1294, 1297 (Fed. Cir. 1997). Applicants respectfully submit that the recited steps for modifying each identified selected failing circuit are not found in the art.

The test for obviousness is what the combined teachings of the references would have suggested to one of ordinary skill in the art. See In re Young, 927 F.2d 588, 591, 18 USPQ2d 1089, 1091 (Fed. Cir. 1991).

The motivation, suggestion or teaching may come explicitly from statements in the prior art, the knowledge of one of ordinary skill in the art, or, in some cases the nature of the problem to be solved. In re Kotzab, 217 F.3d 1365, 1370, 55 USPQ2d 1313, 1317 (Fed. Cir. 2000). Hindsight in impermissible when an examiner rejects an application in reliance upon teachings not drawn from any prior art disclosure, but from the applicant's own disclosure. In re Dembiczak, 175 F.3d 994, 998, 50 USPQ2d 1614, 1616 (Fed. Cir. 1999). Broad conclusory statements standing alone are not "evidence." Id.

The ultimate determination of whether an invention would have been obvious under 35 USC § 103(a) is a legal conclusion based on underlying findings of fact. In re Kotzab, 217 F.3d 1365, 1370, 55 USPQ2d 1313, 1317 (Fed. Cir. 2000). The claimed

invention is unpatentable if the differences between it and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art.

In a proper obviousness determination, "whether the changes from the prior art are 'minor', . . . the changes must be evaluated in terms of the whole invention, including whether the prior art provides any teaching or suggestion to one of ordinary skill in the art to make the changes that would produce the patentee's . . . device." This includes what could be characterized as simple changes, see In re Gordon and Sutherland, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1983).

Determining whether a claim is obvious requires a comparison of the claimed invention including all its limitations with the teaching of the prior art. The total teachings of Bickle, Rosenberg, Dreyer, Carter and Hawley fail to achieve or provide the claimed invention, as taught and claimed by Applicants. Applicants respectfully submit that the subject matter of independent claim 1 is novel and is not rendered obvious by the references of record.

The total teachings of the references of record including Bickle, Rosenberg, Dreyer, Carter and Hawley, considered together in combination, do not provide any remote suggestion and fail to enable one of skill in the art to achieve of the recited subject matter of independent claim 1.

The method for implementing breakpoint based performance measurement using a set of hardware counters for counting hardware events recited in claim 1 includes narrowly defined steps for inserting said start breakpoint instruction and

said stop breakpoint instruction in compiler-generated hardware instructions for a user source code. Claim 1 recited executing said compiler-generated hardware instructions and suspending processing of said hardware instructions responsive to executing said start breakpoint instruction; and responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions; and said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction. Carter discloses source code, a compiler that generates object code, and a hook function call used for the operating system to hand control over to a debugger, and perform debugging operations (blocks 54, 56, 60 of FIG. 5). The above claimed steps of the invention are not taught or suggested by Carter when considered alone and when considered together in combination with all the references of record. Applicants respectfully submit that one of ordinary skill in the art would not have arrived at the presently claimed subject matter without Applicants' teaching of the invention.

A prior art reference may be considered to teach away when "a person of ordinary skill, upon reading the reference, would be discouraged from following the path set out in the reference, or would be led in a direction divergent from the path that was taken by the applicant." In re Gurley, 27 F.3d 551, 553, 31 USPQ2D 1130, 1131 (Fed. Cir. 1994). The above-recited limitations of independent claim 1 also are not shown nor suggested in total combination of Bickle, Rosenberg, Dreyer, Carter and Hawley.

It is impermissible to use the inventor's disclosure as a "road map" for selecting and combining prior art disclosures. In Interconnect Planning Corp. v. Feil 774 F.2d 1132, 1143, 227 USPQ 542, 551 (Fed. Cir. 1985), the Federal Circuit noted, "The invention must be viewed not with the blueprint drawn by the inventor, but in the state of the art that existed at the time." Applicants respectfully submit that without Applicants' own teachings, the invention would not be achieved.

The prior art of record provides no teaching, suggestion or inference in the prior art as a whole or knowledge generally available to one having ordinary skill in the art to achieve the claimed invention, when the invention as claimed is considered "as a whole" as required by 35 U.S.C. § 103 when considering whether the invention would have been obvious when it was made. Graham v. John Deere, 383 U.S. 1, 148 USPQ 459, 472 (1966).

In the words of the Court of Appeals for the Federal Circuit, "The mere fact that the prior art may be modified in the manner suggested by the Examiner does not make the modification obvious unless the prior art suggested the desirability of the modification." In re John R. Fritch, 972 F.2d 1260, 1266, 23 USPQ2d 1780 (Fed. Cir. 1992). See In re Gordon and Sutherland, 733 F.2d 900, 221 USPQ 1125, 1127 (Fed. Cir. 1984), Carl Schenck, A.G. v. Nortron Corp., 713 F.2d 782, 787, 218 USPQ 698, 702 (Fed. Cir. 1983), and In re Sernaker, 702 F.2d 989, 995-96, 217 USPQ 1, 6-7 (Fed. Cir. 1983).

Applicant respectfully submits that the prior art descriptions of Bickle, Rosenberg, Dreyer, Carter and Hawley falls short of applicant's invention, and the

subject matter of the claimed invention as recited in claim 1 would not have been obvious to one of ordinary skill in the art.

Thus, independent claim 1 is patentable.

Claim 4 is patentable

Dependent claim 4 is patentable for the same reasons as claim 1. Dependent claim 4 further recites that the method for implementing breakpoint based performance measurement of claim 1 further includes the step of inserting said start breakpoint instruction and said stop breakpoint instruction at arbitrary user defined locations in said hardware instructions.

The references of record do not suggest the subject matter of the invention as recited by claim 4. In Re Fritch 972 F.2d at 1266, 23 USPQ2d at 1780 (Fed. Cir. 1992), states: "[I]t is impermissible to use the claimed invention as an instruction manual or 'template' to piece together the teachings of the prior art so that the claimed invention is rendered obvious. ... This court has previously stated that '[o]ne cannot use hindsight reconstruction to pick and choose among isolated disclosures in the prior art to deprecate the claimed invention." Applicants respectfully submit that the total teaching of Bickle, Rosenberg, Dreyer, Carter and Hawley would not achieve the claimed invention as recited by claim 4.

Bickle, Rosenberg, Dreyer, Carter and Hawley considering the total teachings in combination, fail to suggest inserting said start breakpoint instruction and said stop breakpoint instruction at arbitrary user defined locations in said hardware instructions, as required by dependent claim 4

Applicant respectfully submits that no teaching or motivation exists for one of ordinary skill in the art to modify the prior art description of Bickle, Rosenberg, Dreyer, Carter and Hawley to achieve the subject matter of the claimed invention as recited in claim 4. Only with the use of applicant's own disclosure, rather than relying upon teachings drawn from any prior art disclosure, would the prior art be modified in the manner suggested by the Examiner. The claimed invention is not rendered obvious by the mere fact that the prior art could be modified in the manner suggested by the Examiner. The prior art fails to suggest the desirability of the compiler-generated hardware instructions (as recited in claim 1), inserting said start breakpoint instruction and said stop breakpoint instruction at arbitrary user defined locations in said hardware instructions (as recited in claim 4), executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions and the breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction (as recited in claim 1). No teaching, suggestion, or motivation is provided by Bickle, Rosenberg, Dreyer, Carter and Hawley and no general knowledge in the art exists to achieve the subject matter of the invention, as claimed by Applicants in dependent claim 4.

Thus, dependent claim 4 is patentable.

Claim 6 is patentable

Independent claim 6 is patentable generally for the same reasons as set forth above with respect to the method claim 1.

Independent claim 6 recites apparatus for implementing breakpoint based performance measurement comprising: a plurality of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and cache misses; a source level debugger including a breakpoint manager; said breakpoint manager including a performance measurement program and a user interface; said breakpoint manager utilizing said performance measurement program and said user interface for defining a set of said hardware counters for counting user specified programmable processor events and for inserting a start breakpoint instruction and a stop breakpoint instruction in hardware instructions; said breakpoint manager enabling a user to specify a start bound and an end bound of a performance collection region of a user source code and said set of hardware counters; a compiler generating hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction and inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code.

Applicants respectfully submit that the above recited limitations defining the subject matter of independent claim 6 is not rendered obvious by the total teachings of the references of record including Bickle, Rosenberg, Dreyer, Carter and Hawley. No

teaching, suggestion, or motivation provided by Bickle, Rosenberg, Dreyer, Carter and Hawley nor any general knowledge in the art exists to achieve the breakpoint manager including a performance measurement program and a user interface, nor the compiler generating hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction and inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code defining the subject matter of the invention, as claimed by Applicants in independent claim 6.

Independent claim 6 further recites user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions responsive to executing said start breakpoint instruction and generating a processor interrupt for entering interrupt handler instructions and for calling said breakpoint manager; said breakpoint manager for generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction; and said user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction.

Applicants submit that the subject matter of the claimed invention as recited in claim 6 would not have been obvious to one of ordinary skill in the art in view

of the references of record. No hint is found in the references of record and the references of record do not suggest said breakpoint manager for generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction, as taught and claimed by Applicants in claim 6. There is nothing in the applied references which would have motivated one of skill in the art to provide user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions responsive to executing said start breakpoint instruction and generating a processor interrupt for entering interrupt handler instructions and for calling said breakpoint manager; said breakpoint manager for generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction; and said user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction.

There is no teaching or suggestion in any of the cited references, individually or taken as a whole, to make the claimed invention obvious.

Thus, claim 6 is patentable.

Claim 11 is patentable

Independent claim 11 is patentable for the same reasons as claim 1. Independent claim 11 recites a computer program product for implementing breakpoint based performance measurement in a computer system, said computer program product including instructions executed by the computer system to cause the computer system to perform the steps of: defining a set of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and translation lookaside buffer misses; providing compiler-generated hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction; inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code; executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions responsive to executing said start breakpoint instruction; responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions; said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction; executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions and stopping said defined set of hardware counters,

responsive to executing said stop breakpoint instruction; and receiving user selections for a start bound and an end bound of a performance collection region of said user source code and said set of hardware counters.

The references of record do not suggest the subject matter of the invention as recited by claim 11. The references of record do not suggest the recited steps performed by the computer system of the invention as recited by claim 11.

Applicants submit that no sufficient reason exists so that one skilled in the art would have arrived at the claimed subject matter in considering the total teachings of the references of record.

The Examiner acknowledges that Bickle does not disclose suspending processing of said hardware instructions responsive to executing said start breakpoint instruction. Rosenberg and Dreyer also fail to suggest executing said hardware instructions and suspending processing of said hardware instructions responsive to executing said start breakpoint instruction.

The Examiner further cites Carter reference and maintains that that Carter teaches that compilers generate hardware instructions. Applicants respectfully submit that the teachings of Carter are different from and fail to achieve executing said hardware instructions and suspending processing of said hardware instructions responsive to executing said start breakpoint instruction, as taught and claimed by Applicants. The compiler of Carter generates object code; however Applicants respectfully submit that Carter does not teach generating hardware instructions as taught and claimed by Applicants, when the claim term is construed in a manner

consistent with the ordinary and customary meaning of the claim term and as taught by Applicants the patent specification where the invention is described.

The claimed features and limitations defined by claim 11 are not shown nor suggested in total combination of the references of record. Claims should be given their broadest reasonable interpretation consistent with the specification. The claim language itself governs its meaning. Vitronics Corp. v. Conceptronic, Inc., 90 F.3d 1576, 1582 (Fed. Cir. 1996). The meaning of claim language is construed according to its usage and context. ResQNet.com, Inc. v. Lansa, Inc., 346 F.3d 1374, 1378 (Fed. Cir. 2003). The touchstone for discerning the usage of claim language is the understanding of those terms among artisans of ordinary skill in the relevant art at the time of invention. See Rexnord Corp. v. Laitram Corp., 274 F.3d 1336, 1342 (Fed. Cir. 2001). Indeed, normal rules of usage create a "heavy presumption" that claim terms carry their accustomed meaning in the relevant community at the relevant time. CCS Fitness, Inc. v. Brunswick Corp., 288 F.3d 1359, 1366 (Fed. Cir. 2002) (citing Johnson Worldwide Assocs., Inc. v. Zebco Corp., 175 F.3d 985, 989 (Fed. Cir. 1999)). The best source for discerning the proper context of claim terms is the patent specification wherein the patent applicant describes the invention. In addition to providing contemporaneous technological context for defining claim terms, the patent applicant may also define a claim term in the specification "in a manner inconsistent with its ordinary meaning." Boehringer Ingelheim Vetmedica, Inc. v. Schering-Plough Corp., 320 F.3d 1339, 1347 (Fed. Cir. 2003) (citing Teleflex, 299 F.3d at 1325-26). Dictionaries, encyclopedias and treatises are particularly useful resources to assist in determining the

ordinary and customary meanings of claim terms. Tex. Digital Sys., Inc. v. Telegenix, Inc., 308 F.3d 1193, 1202 (Fed. Cir. 2002).

It is impermissible to use the inventor's disclosure as a "road map" for selecting and combining prior art disclosures. In Interconnect Planning Corp. v. Feil 774 F.2d 1132, 1143, 227 USPQ 542, 551 (Fed. Cir. 1985), the Federal Circuit noted that "The invention must be viewed not with the blueprint drawn by the inventor, but in the state of the art that existed at the time." Rejections based on § 103 must rest on a factual basis with these facts being interpreted without hindsight reconstruction of the invention from the prior art. The Examiner may not, because of doubt that the invention is patentable, resort to speculation, unfounded assumption or hindsight reconstruction to supply deficiencies in the factual basis for the rejection. See In re Warner, 379 F.2d 1011, 1017, 154 USPQ 173, 178 (CCPA 1967), cert. denied, 389 U.S. 1057 (1968).

Applicants respectfully submit that one of ordinary skill in the art would not have been led to the claimed invention as recited in independent claim 11 by the reasonable teachings and suggestions found in the prior art, including the Bickle, Rosenberg, Dreyer et al., Carter et al., and Hawley references, when considered with the knowledge generally available to one of ordinary skill in the art.

The rejection of the claim 11 under 35 USC §103 is incorrect and should be reversed.

Independent claim 11 is patentable over the prior art.

D. THE REJECTION OF CLAIM 7 UNDER 35 USC 103 SHOULD BE REVERSED

The Board should reverse the rejection of the claim 7 under 35 USC §103 over Bickle, Rosenberg, Dreyer, Carter and Hawley.

Claim 7 is patentable

Dependent claim 7 further defines the apparatus for implementing breakpoint based performance measurement of the invention as recited in independent claim 6, and is patentable for the same reasons as claim 6. Dependent claim 7 further recites that said start breakpoint instruction includes encoded information specifying said defined set of hardware counters.

Applicant respectfully submits that no teaching or motivation exists for one of ordinary skill in the art to modify the prior art description of Bickle, Rosenberg, Dreyer, Carter and Hawley to achieve the subject matter of the claimed invention as recited in claim 7. Hawley teaches analyzing the breakpoint instruction to determine the type of breakpoint that was encountered.

Bickle, Rosenberg, Dreyer, Carter and Hawley considering the total teachings in combination, fail to suggest that a start breakpoint instruction includes encoded information specifying said defined set of hardware counters, as required by dependent claim 7

Only with the use of applicant's own disclosure, rather than relying upon teachings drawn from any prior art disclosure, would the prior art be modified in the manner suggested by the Examiner.

Thus, dependent claim 7 is patentable.

E. CONCLUSION

Claims 1-2, 4-11, 13, and 14 are patentable over all the references of record and are not rendered obvious by the prior art. Each of the claims 1-2, 4-11, 13, and 14 is patentable and the Examiner's rejections should be reversed.

It is respectfully requested that the final rejection be reversed.

Respectfully submitted,

S-signature by

___/Joan Pennington/_____
By: Joan Pennington
Reg. No. 30,885
Telephone: (312) 670-0736

April 24, 2008

(8) CLAIMS APPENDIX

CLAIMS ON APPEAL

1. A method for implementing breakpoint based performance measurement using a set of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and cache misses; said method comprising:

providing compiler-generated hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction;

inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code;

executing said compiler-generated hardware instructions and suspending processing of said hardware instructions responsive to executing said start breakpoint instruction;

responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions;

said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction;

executing said compiler-generated hardware instructions and suspending

processing of said compiler-generated hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction; and

providing a debugger breakpoint manager including a performance measurement program and a user interface, and enabling a user to specify a start bound and an end bound of a performance collection region of said user source code and said set of hardware counters.

2. A method for implementing breakpoint based performance measurement as recited in claim 1 wherein said predefined processor events further include at least one of processor instructions executed, a defined type of processor instruction executed, and translation lookaside buffer misses.

4. A method for implementing breakpoint based performance measurement as recited in claim 1 wherein the inserting step includes inserting said start breakpoint instruction and said stop breakpoint instruction at arbitrary user defined locations in said hardware instructions.

5. A method for implementing breakpoint based performance measurement as recited in claim 1 includes the steps of enabling a user to interrogate a program state and to request said start processing instruction.

6. Apparatus for implementing breakpoint based performance measurement comprising:

a plurality of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and cache

misses;

a source level debugger including a breakpoint manager;

said breakpoint manager including a performance measurement program and a user interface;

said breakpoint manager utilizing said performance measurement program and said user interface for defining a set of said hardware counters for counting user specified programmable processor events and for inserting a start breakpoint instruction and a stop breakpoint instruction in hardware instructions;

said breakpoint manager enabling a user to specify a start bound and an end bound of a performance collection region of a user source code and said set of hardware counters;

a compiler generating hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction and inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code;

user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions responsive to executing said start breakpoint instruction and generating a processor interrupt for entering interrupt handler instructions and for calling said breakpoint manager;

said breakpoint manager for generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware

instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction; and

said user program means for executing said compiler-generated hardware instructions and suspending processing of the hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction.

7. Apparatus for implementing breakpoint based performance measurement as recited in claim 6 wherein start breakpoint instruction includes encoded information specifying said defined set of hardware counters.

8. Apparatus for implementing breakpoint based performance measurement as recited in claim 6 wherein said breakpoint manager, responsive to said start breakpoint instruction, records user information specifying said defined set of hardware counters.

9. Apparatus for implementing breakpoint based performance measurement as recited in claim 6 wherein said predefined processor events further include at least one of processor instructions executed, a defined type of processor instruction executed, and translation lookaside buffer misses.

10. Apparatus for implementing breakpoint based performance measurement as recited in claim 6 wherein said breakpoint manager inserts said start breakpoint instruction and said stop breakpoint instruction at arbitrary user defined locations in said hardware instructions.

11. A computer program product for implementing breakpoint based performance measurement in a computer system, said computer program product including instructions executed by the computer system to cause the computer system to perform the steps of:

defining a set of hardware counters for counting hardware events; said hardware counters being programmable for counting predefined programmable processor events; said predefined programmable processor events including processor cycles and translation lookaside buffer misses;

providing compiler-generated hardware instructions defining breakpoint instructions within an instruction stream; said compiler-generated hardware instructions including a start breakpoint instruction and a stop breakpoint instruction;

inserting said start breakpoint instruction and said stop breakpoint instruction in compiler-generated hardware instructions for a user source code;

executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions responsive to executing said start breakpoint instruction;

responsive to executing said start breakpoint instruction generating a processor interrupt for entering interrupt handler instructions and calling breakpoint instructions;

said breakpoint instructions generating a start processing instruction to return processing from said interrupt handler instructions to said compiler-generated hardware instructions and starting said defined set of hardware counters, responsive to said generated start processing instruction;

executing said compiler-generated hardware instructions and suspending processing of said compiler-generated hardware instructions and stopping said defined set of hardware counters, responsive to executing said stop breakpoint instruction; and

receiving user selections for a start bound and an end bound of a performance collection region of said user source code and said set of hardware counters.

13. A computer program product for implementing breakpoint based performance measurement as recited in claim 11 wherein the inserting step includes inserting said start breakpoint instruction and said stop breakpoint instruction at arbitrary user defined locations in said hardware instructions.

14. A computer program product for implementing breakpoint based performance measurement as recited in claim 11 wherein said predefined processor events include at least one of processor instructions executed, cache misses, and a defined type of processor instruction executed.

(9) EVIDENCE APPENDIX

None.

(10) RELATED PROCEEDINGS APPENDIX

None.